Single machine parallel-batch scheduling with deteriorating jobs

Xianglai Qi*, Shiguo Zhou, Jinjiang Yuan

Department of Mathematics, Zhengzhou University, Zhengzhou, Henan 450052, People's Republic of China

Abstract

We consider several single machine parallel-batch scheduling problems in which the processing time of a job is a linear function of its starting time. We give a polynomial-time algorithm for minimizing the maximum cost, an $O(n^5)$ time algorithm for minimizing the number of tardy jobs, and an $O(n^2)$ time algorithm for minimizing the total weighted completion time. Furthermore, we prove that the problem for minimizing the weighted number of tardy jobs is binary NP-hard.

© 2009 Published by Elsevier B.V.

1. Introduction

Traditional scheduling problem assumes that the processing time of a given job is fixed. However, the processing times may change in the real world. Examples can be found in steel production, fire fighting and maintenance, etc., where any delay in processing a task may increase its completion time. The reader is referred to [13,19] for other examples.

Gupta and Gupta [11], and Browne and Yechiali [5] first introduced deteriorating jobs, i.e., jobs whose processing times is an increasing function of their starting time. They assumed that the actual processing time is $a_j + b_j t$ ($b_j > 0$), where $a_j$ is the basic processing time, $b_j$ the deteriorating rate, and $t$ the starting time, of job $J_j$. They showed that sequencing the jobs in non-decreasing order of $\{a_j\}$ minimizes the makespan. Mosheiov [16] considered the total flow-time minimization problem with the actual processing time of job $J_j$ equal to $a + b_j t$, where $a$ is a common basic processing time. He showed that an optimal schedule is V-shaped with respect to $b_j$. Bachman and Janiak [2] and Bachman, Janiak and Kovalyov [3] proved that the problems of minimizing maximum lateness and total weighted completion time are NP-hard. Mosheiov [17] introduced the simple linear deteriorating jobs, where all jobs have a basic processing time equal to 0. He presented polynomial-time algorithms for minimizing makespan, flow-time, total weighted flow time, maximum lateness, maximum tardiness and the number of tardy jobs. Cheng and Ding [8] studied the step-deteriorating model and gave a pseudo-polynomial-time algorithm for minimizing makespan. Kononov and Gawiejnowicz [12] considered the dedicated machine problems with deteriorating jobs. Recently, Barketau et al. [4] considered the problem of scheduling $n$ jobs on a single machine, where the jobs are processed in serial batches and the processing time of each job is a step function depending on its waiting time in processing procedure of the batch containing the job. They showed that the problem is NP-hard in the strong sense even if all $b_i$ are equal, NP-hard even if $b_i = a_i$ for all $i$, and non-approximable in polynomial time with a constant performance guarantee $\Delta < 3/2$, unless $P = NP$. Other results of scheduling models considering deterioration effect can be found in Wang et al. [20–24]. An extensive survey of different models and problems was provided by Alidaee and Womer [1]. Cheng, Ding and Lin [9] later presented an updated survey of the results on scheduling problems with time-dependent processing.

* Research partially supported by NSFC (10671183), NFSC-RGC (70731160633) and SRFDP (20070459002).
* Corresponding author. Tel.: +86 013938580082.
E-mail address: qx1@zzu.edu.cn (X. Qi).

0304-3975/$– see front matter © 2009 Published by Elsevier B.V.
doi:10.1016/j.tcs.2008.11.009
times. In most of these achievements, the positive results are usually obtained by dynamic programming algorithms and the dominated properties of jobs in an optimal schedule (such as, SPT-property, EDD-property, LDT-property, and V-shaped property), and the negative results (NP-hardness) are usually obtained by reductions from Partition Problem, 3-Partition Problem or Subset Product Problem.

By Brucker et al. [6], a parallel-batch processing machine is a machine that can process up to \( b \) jobs simultaneously as a batch. The time the machine takes to process a batch is given by the maximum processing time of all jobs contained in the batch, and once processing of a batch is initiated, it cannot be interrupted, nor can other jobs be introduced into the batch. Such a processing environment arises in the manufacturing industry, and is called parallel-batch scheduling. As an example, the final stage in the production of VLSI circuits is a burn-in operation in which the chips are put into an oven in batches and heated for a prolong period (in terms of days) in order to bring out any latent defect. Different types of chips have different minimum baking times. That is, they are allowed to stay in the oven for longer than the prescribed period but not for shorter. Due to this application, the above model is often called the burn-in model, see [14] for more details of the background. An extensive survey of different models and problems was provided by Potts and Kovalyov [18]. By [6], the capacity \( b \) has two variants: the unbounded model, in which \( b \geq n \) so that there is effectively no upper bound on the number of jobs that can be processed in the same batch; and the bounded model, in which \( b \) is a constant smaller than \( n \) so that there is a restrictive upper bound, where \( n \) is the number of jobs.

Although the topics of deteriorating jobs and parallel-batch processing machine have been widely investigated in scheduling research, to the best of our knowledge, no work has been done on model combining both aspects. But, job deterioration and parallel-batch processing co-exist in many realistic scheduling situations. Example can be found in steel production. In this paper, we study the scheduling model with deteriorating jobs and parallel-batch machine.

In this paper, for the new model, we give a characterization of a class of optimal schedules, which forms the basis of polynomial dynamic programming algorithms for specific cost functions. We present a polynomial-time algorithm for minimizing the maximum cost, an \( O(n^2) \) time algorithm for minimizing the number of tardy jobs, and an \( O(n^2) \) time algorithm for minimizing the total weighted completion time. Furthermore, we prove that minimizing the weighted number of tardy jobs is NP-hard.

The remainder of this paper is organized as follows. We present in the next section a description of the new model, introduce our notation. Then we give our dynamic programming algorithms for polynomially solvable problems in Section 3 and proof for NP-hard problem in Section 4. Finally, we conclude the paper.

2. Problem description and notation

The scheduling model that we analyze is as follows. There are \( n \) independent jobs, \( J = \{J_1, J_2, \ldots, J_n\} \), which are simultaneously available at time \( t_0 > 0 \), i.e., \( r_j \equiv t_0 \) (\( 1 \leq j \leq n \)), to be scheduled non-preemptively on a single parallel-batching machine. We assume, as in Mosheiov [17] and Chen [7], that the actual processing time of a job \( J_j \) is \( p_j = b_j t \), where \( t \) and \( b_j \) are the starting time and the growth (or deterioration) rate of \( J_j \), respectively. The assumption “\( t_0 > 0 \)” is made here to avoid the trivial case of \( t_0 = 0 \) (when \( t_0 = 0 \), the completion time of each job will be 0). Also, each job has a cost function \( f_j \), where \( f_j(t) \geq 0 \) denotes the cost incurred if the job is completed at time \( t \). Throughout this paper, we consider only regular cost functions, i.e. we assume that \( f_j(t) \) is a non-decreasing function of \( t \), for \( j = 1, \ldots, n \). Sometimes, each job \( J_j \) has a due date \( d_j \geq 0 \) by which it should ideally be completed, a deadline \( d_j \), by which it must be completed, and a weight \( w_j \), which is a measure of its importance; when there is ambiguity, we state explicitly whenever due dates, or weights are present. The weights and due dates are typically used to define cost objective functions; the deadlines restrict the availability of jobs.

The batching machine is available from time \( t_0 \) onwards and can process up to \( b \) jobs simultaneously. The jobs are processed in parallel batches. The processing time of a batch is equal to the largest processing time of any job in the batch. The completion time of all jobs in a batch is defined as the completion time of the batch. Without loss of generality, we assume that the job parameters are integral, unless stated otherwise.

For problems of minimizing a regular objective function with job release dates \( r_j \equiv t_0 \) (\( 1 \leq j \leq n \)), we can easily see that there must be an optimal schedule in which the batches are processed continuously from time \( t_0 \) onwards. Throughout the paper, we restrict attention to schedules with this property. Thus, a schedule \( \sigma \) can be simply denoted a sequence of batches \( \sigma = (B_1, \ldots, B_r) \), where each batch \( B_l \) (\( l = 1, \ldots, r \)) is a set of jobs. We use \( S(B_l) = S(B_l, \sigma) \) and \( C(B_l) = C(B_l, \sigma) \) to denote the starting time and completion time of a batch \( B_l \) in a schedule \( \sigma \). Hence, the processing time of a batch \( B_l \) is \( P(B_l) = \max\{b_j : J_j \in B_l\} \times S(B_l) \) and the completion time \( C(B_l) = S(B_l) + P(B_l) = (1 + \max\{b_j : J_j \in B_l\}) \times S(B_l) \). Note that the completion time of job \( J_j \) in \( \sigma \), for each \( J_j \in B_l \) and \( l = 1, \ldots, r \), is \( C_j(\sigma) = C(B_l) \). When there is no ambiguity, we abbreviate \( C_j(\sigma) \) to \( C_j \).

In this paper, we consider the unbounded model, in which \( b \geq n \) so that there is effectively no upper bound on the number of jobs that can be processed in the same batch. Following [6,15,17], we call this model the parallel-batch scheduling problem under simple linear deterioration and denote it by

\[
1|p\text{-batch}; r_j = t_0; p_j = b_j t f_j.
\]

where “p-batch” means parallel batch, and \( f \) is a regular objective function, to be minimized.

The aim is to minimize the scheduling cost, measured either by a regular minmax objective function \( f_{\text{max}} = \max_{1 \leq j \leq n}\{f_j(C_j)\} \), or by a regular min-sum objective function \( \sum_{j=1}^{n} f_j = \sum_{j=1}^{n} f_j(C_j) \). Specific regular objective functions
that we consider are the makespan $C_{\text{max}}$, defined as $C_{\text{max}} = \max_{1 \leq j \leq n} \{ C_j \}$; maximum lateness $L_{\text{max}}$, defined as $L_{\text{max}} = \max_{1 \leq j \leq n} \{ C_j - d_j \}$; total weighted completion time $\sum_{j=1}^{n} w_j C_j$; total weighted tardiness $\sum_{j=1}^{n} w_j T_j$, where $T_j = \max \{ C_j - d_j, 0 \}$; and weighted number of tardy jobs $\sum_{j=1}^{n} w_j U_j$, where $U_j$ is 0−1 indicator variable that takes the value 1 if $j$ is tardy, i.e., if $C_j > d_j$, and the value 0 if $j$ is on time, i.e., if $C_j \leq d_j$. We also provide results for the unweighted versions of these minsum objective functions in which $w_j = 1$ for $j = 1, \ldots, n$.

3. Polynomially-time solvable problems

In Section 3.1, we first give three basic lemmas, which are useful for our dynamic programming algorithms.

3.1. Three basic lemmas

Since the objective function $f$ is regular, we have the following result about the optimal solutions which is similar to Lemma 1 in Brucker et al. [6] and Lemma 2.1.1 in Yuan et al. [25].

**Lemma 3.1.1.** For every regular objective function $f$, there is an optimal batch sequence $(B_1, B_2, \ldots, B_r)$ for problem 1|p-batch; $r_j = t_0; p_j = b_j t | f$ such that, for every two batches $B_x$ and $B_y$ with $x < y$,

$$\max \{ b_j : j \in B_x \} < \min \{ b_j : j \in B_y \}.$$  

**Proof.** Let $\sigma$ be an optimal schedule for problem 1|p-batch; $r_j = t_0; p_j = b_j t | f$, where $\sigma = (B_1, B_2, \ldots, B_r)$. Suppose that there are two batches $B_x$ and $B_y$ in $\sigma$ with $x < y$ such that $\max \{ b_j : j \in B_x \} \geq \min \{ b_j : j \in B_y \}$. Then there is $j \in B_y$ such that $b_j \leq \max \{ b_j : j \in B_x \}$. By shifting $j$ from $B_y$ to $B_x$, we obtain a new batch sequence $\sigma^* = (B_1', B_2', \ldots, B_r')$. Note that, if $B_y = (j_1, j_2)$, then $\sigma^*$ will not appear in $\sigma^*$. Since $b_j \leq \max \{ b_j : j \in B_x \}$, we have that $p(B_x \cup j) = p(B_x) + p(B_y \setminus \{ j \}) \leq p(B_y)$. Furthermore, the completion time of job $j$ decreases from $C(B_y)$ to $C(B_x)$, while because the starting times of the other jobs do not increase, the completion times also do not increase. Since $f$ is regular, $\sigma^*$ is still an optimal schedule. A finite number of repetitions of this procedure yields an optimal schedule of the required form. □

In the remaining part of this section, we assume that the jobs have been re-indexed according to the shortest deterioration rate (SDR) rule so that $b_1 \leq b_2 \leq \cdots \leq b_n$. We refer to a schedule which satisfies the property in Lemma 3.1.1 an SDR-batch schedule. By Lemma 3.1.1, we only need to find an optimal SDR-batch schedule. Then an SDR-batch schedule can be defined as $(B_1, \ldots, B_r)$, in which each batch $B_x (1 \leq x \leq r)$ is of the form

$$B_x = \{ j_1, j_{k+1}, \ldots, j_{l(x+1)-1} \} \quad (1 = l_1 < l_2 < \cdots < l(r+1) = n + 1).$$

To simplify the discussion, let $\mathcal{U} = \{ j_{2}, j_{3}, \ldots, j_{l(r+1)-1} = n \}$ for an SDR-batch schedule. Note that each job $j_{l(x+1)-1}$ in $\mathcal{U}$ has the largest deterioration rate in batch $B_x$, and the processing time of batch $B_x$ is $P(B_x) = b_{l(x+1)-1} S(B_x)$. We can regard the deterioration rate $b_{l(x+1)-1}$ of job $j_{l(x+1)-1}$ as the deterioration rate of batch $B_x$.

For 1|p-batch; $r_j = t_0; p_j = b_j t | C_{\text{max}}$, we have the following lemma.

**Lemma 3.1.2.** The maximum completion time of the jobs subset $S$ is $C_{\text{max}}(S) = t_0 \prod_{i \in S} (1 + b_i), \forall S \subseteq \mathcal{J}$.

So we have the similar lemma for 1|p-batch; $r_j = t_0; p_j = b_j t | C_{\text{max}}$:

**Lemma 3.1.3.** For an SDR-batch schedule $(B_1, \ldots, B_r)$, the maximum completion time of jobs is $C_{\text{max}} = t_0 \prod_{j \in \mathcal{U}} (1 + b_i)$.

We also have the following observation.

**Observation 3.1.4.** For a given batch schedule $\sigma$, the completion time of every job is proportional to the starting time $t_0$ of $\sigma$.

3.2. 1|p-batch; $r_j = t_0; p_j = b_j t | f_{\text{max}}$

Note that the problem of minimizing the makespan is solved trivially by putting all jobs in one batch $B_1$. The minimum makespan is then $C_{\text{max}} = p(B_1) = t_0 = t_0(1 + b_0)$.

The decision version of problem 1|p-batch; $r_j = t_0; p_j = b_j t | f_{\text{max}}$, denoted by

$$1|p\text{-batch;} \quad r_j = t_0; \quad p_j = b_j t \quad | f_{\text{max}} \leq \gamma,$$  

(1)

asks whether there is a feasible schedule $\sigma$ such that $f_{\text{max}}(\sigma) \leq \gamma$, i.e., $f_i(C_i(\sigma)) \leq \gamma$, for each job $j$. By Lemma 3.1.3, in any schedule, the completion time of every job cannot exceed $D = t_0 \prod_{1 \leq j \leq n} (1 + b_i)$. Hence, the value of $Y$ can be chosen as an integer in the interval $[0, \Delta]$, where $\Delta = \max_{1 \leq j \leq n} f_i(D)$. By the binary search method for the value of $Y \in [0, \Delta]$, we can obtain the following observation.

**Observation 3.2.1.** If, for each integer $Y \in [0, \Delta]$, decision problem (1) can be solved in $O(F(n))$ time, then problem 1|p-batch; $r_j = t_0; p_j = b_j t \ | f_{\text{max}}$ can be solved in $O(F(n) \log \Delta)$ time.

Suppose that $\Delta$ is polynomially bounded in the size of the input. This means that, if the decision problem is solvable in polynomial time, then minimizing $f_{\text{max}}$ is solvable in polynomial time. The decision question “is $f_{\text{max}} \leq \gamma$?” can be answered in polynomial time as follows.
Define $d_i = \max\{C_i : C_i$ is an integer with $f_i(C_i) \leq Y\}$, for each job $J_i$. Clearly, each $d_i$ can be calculated by the binary search method for the value of $C_i \in [0, D]$ in $O(\log D)$ time. This means that, for any given $Y$, we can determine all value of $d_i$, $1 \leq i \leq n$, in $O(n \log D)$ time. And note that an upper bound on the total size of $D$ under a binary encoding is $\log_2 t_0 + n \log_2(1 + b_i)$, which is polynomially bounded in $n$, $\log_2 t_0$ and $\log_2 b_n$.

The above discussion means the following result.

**Theorem 3.2.2.** By using $O(n \log D)$ time, decision problem (1) is polynomially reduced to problem 1$p$-batch; $r_j = t_0$; $p_j = b_j t |l_{\max} \leq 0$.

We now focus our attention to problem 1$p$-batch; $r_j = t_0$; $p_j = b_j t |l_{\max} \leq 0$.

First consider, for a given $j$ with $1 \leq j \leq n$, the jobs in $J_j = \{J_{j1}, J_{j2}, \ldots, J_{jn}\}$ to be processed in the batching machine under simple linear deterioration. If $\sigma$ is an SDR-batch schedule for the jobs in $J_j$ such that $l_{\max}(\sigma) \leq 0$, we say that $\sigma$ is $J_j$-feasible. Define, for a given $j$ with $1 \leq j \leq n$, $F(j) = \min\{C_j(\sigma) : \sigma$ is a $J_j$-feasible schedule $\}$. If there is no $J_j$-feasible schedules we define $F(j) = \infty$. We further define $F(0) = t_0$ as initial condition. If $\sigma$ is an $J_j$-feasible SDR-batch schedule such that the last batch is of the form $[J_{j1}, \ldots, J_{jk}]$, then the completion time of job $J_j$ equals $(1 + b_j)$ times the completion time of $J_{jk}$.

Hence, $F(j)$ can be calculated by the following dynamic programming recursion:

$$F(j) = (1 + b_j) \min_{0 \leq i < j} \{F(i) + \delta(i, j)\},$$

where

$$\delta(i, j) = \begin{cases} 0, & \text{if } (1 + b_j)F(i) \leq d_i \text{ for } i + 1 \leq l \leq j, \\ \infty, & \text{otherwise}. \end{cases}$$

The dynamic programming function has $n + 1$ states. Each recursion runs only $O(n^2)$ time. Hence, all $F(j)$ can be calculated in $O(n^3)$ time.

Problem 1$p$-batch; $r_j = t_0$; $p_j = b_j t |l_{\max} \leq 0$ has a solution if and only if $F(n) < \infty$. We thus have

**Theorem 3.2.3.** Problem 1$p$-batch; $r_j = t_0$; $p_j = b_j t |l_{\max} \leq 0$ can be solved in $O(n^3)$ time.

Combining Observation 3.2.1, Theorems 3.2.2 and 3.2.3, we obtain

**Theorem 3.2.4.** Problem 1$p$-batch; $r_j = t_0$; $p_j = b_j t |f_{\max} \leq 0$ can be solved in $O(n(n^2 + \log D) \log \Delta)$ time.

3.3. 1$p$-batch; $r_j = t_0$; $p_j = b_j t |\sum U_j$.

We first give a dynamic programming algorithm for general sum-form objective function $\sum f_j$.

Let $N = \max\{f_j(t_0) \prod_{i=1}^n (1 + b_i) : 1 \leq j \leq n\}$. Then $f_j(C_j(\sigma)) \leq N$ in any SDR-batch schedule $\sigma$.

Let $\varphi(j)$ be problem 1$p$-batch; $r_j = t_0$; $p_j = b_j t |\sum_{1 \leq i \leq j} f_i$ with jobs $J_{1j}, J_{2j}, \ldots, J_{nj}$, where $1 \leq j \leq n$. Let $H_t(j, t)$ be the minimum completion time of job $J_j$ in an SDR-batch schedule for $\varphi(j)$ under the restriction that objective value of the schedule is exactly $t$. If problem $\varphi(j)$ has no SDR-batch schedule with objective value $t$, we define $H_t(j, t) = \infty$. We further define $H_0(0, 0) = t_0$ and $H_t(0, 0) = \infty$ for any $t > 0$ as initial condition.

If $\sigma$ is an SDR-batch schedule for problem $\varphi(j)$ such that the objective value is $t$ and the completion time of $J_j$ is minimum, then the last batch in $\sigma$ is of the form $[J_i, \ldots, J_j]$ for some $i$ with $1 \leq i \leq j$, and therefore contributes $\sum_{i \leq j} f_i(H_t(i, j))$. Furthermore, the schedule $\pi$ obtained by restricting $\sigma$ in the jobs $J_{i1}, J_{i2}, \ldots, J_{in-1}$ is an SDR-batch schedule for problem $\varphi(i - 1)$ such that the objective value is $t^* = t - \sum_{i \leq j} f_i(H_t(i, j))$. Since $\pi$ is an SDR-batch schedule, the completion time of $J_j$ is calculated by

$$H_t(j, t) = (1 + b_j)H_t(t^*, i - 1),$$

where the value of $t^*$ satisfies the condition

$$t^* + \sum_{i \leq j} f_i((1 + b_j)H_t(t^*, i - 1)) = t.$$

By summing the above discussion, $H_t(j, t)$ can be calculated by the following dynamic programming recursion:

$$H_t(j, t) = (1 + b_j) \min_{i \leq j} \{H_t(t^*, i - 1),$$

where

$$T = \left\{ t^* : t^* + \sum_{i \leq j} f_i((1 + b_j)H_t(t^*, i - 1)) = t \right\}.$$
For any positive integer $p$, if $f_j$ can be solved in $O(n^2N^2)$ time. The minimum objective value of the problem is given by
$$\min\{t \in [0, nN] : H(t, n) < \infty\}.$$ We thus have:

**Theorem 3.3.1.** Problem 1 $|p\text{-batch}; r_j = t_0; p_j = b_jt| \sum f_j$ can be solved in $O(n^2N^2)$ time.

When the objective function is $\sum U_j$, we have $N = 1$ in the above discussion. Hence, we have

**Theorem 3.3.2.** Problem 1 $|p\text{-batch}; r_j = t_0; p_j = b_jt| \sum U_j$ can be solved in $O(n^5)$ time.

Also, when the objective function is $\sum w_jU_j$, we have $N = \max w_j$. So we can derive the below Theorem:

**Theorem 3.3.3.** Problem 1 $|p\text{-batch}; r_j = t_0; p_j = b_jt| \sum w_jU_j$ can be solved in pseudo-polynomial-time.

### 3.4. 1 $|p\text{-batch}; r_j = t_0; p_j = b_jt| \sum w_jC_j$

In this section, we present an $O(n^2)$ time dynamic programming algorithm for minimizing the total weighted completion time $\sum w_jC_j$. Different from the above two dynamic programming algorithms, a backward dynamic programming algorithm is provided. We first derive a very useful characterization of optimal solutions.

**Lemma 3.4.1.** For any positive integer $p$, if $\sigma$ is an optimal schedule for minimizing the total weighted completion time when the starting time of $\sigma$ is $t_0$, it is also an optimal schedule when the starting time is $p\sigma_0$. That is, an optimal schedule is independent on its starting time.

**Proof.** Note that, by Observation 3.1.4, for a given schedule $\sigma$, the completion time of job $J_j$ is $C_j$ when the starting time of $\sigma$ is $t_0$. If and only if the completion time is $pC_j$ when the starting time is $p\sigma_0$. Then the result can be observed. \(\square\)

Let $F(j)$ be the minimum total weighted completion time for SDR-batch schedules containing the last $n - j + 1$ jobs $J_{j+1}, \ldots, J_n$. Processing of the first batch in the schedule starts at time $t_0$. Furthermore, whenever a new batch is added to the beginning of this schedule, there is a corresponding delay in the processing of all batches. Suppose that a batch $J_{k-1}$, which has processing time $t_0(1 + b_{k-1})$, is inserted at the start of a schedule for jobs $J_k, \ldots, J_n$, then the starting time of these jobs increases from $t_0$ to $t_0(1 + b_{k-1})$. By Lemmas 3.1.3 and 3.4.1, the total weighted completion time of jobs $J_k, \ldots, J_n$ increases from $C_j$ to $(1 + b_{k-1})C_j$, $k \leq j \leq n$. Hence, the total weighted completion time of jobs $J_k, \ldots, J_n$ increases from $F(k)$ to $(1 + b_{k-1})F(k)$. We are now ready to give the dynamic programming recursion. The initialization is $F_{n+1} = 0$ and the recursion for $j = n, n-1, \ldots, 1$ is

$$F(j) = \min_{j < k \leq n+1} \left(t_0 \sum_{j \leq i < k-1} w_i(1 + b_{k-1}) + (1 + b_{k-1})F(k)\right)$$

$$= \min_{j < k \leq n+1} \left((1 + b_{k-1}) \sum_{j \leq i < k-1} w_i + F(k)\right).$$

The optimal solution value is then equal to $F_1$, and the corresponding optimal schedule is found by backtracking. Under the most natural implementation, the algorithm requires $O(n^2)$ time. So we have:

**Theorem 3.4.2.** Problem 1 $|p\text{-batch}; r_j = t_0; p_j = b_jt| \sum w_jC_j$ can be solved in $O(n^2)$ time.

### 4. NP-hardness of 1 $|p\text{-batch}; r_j = t_0; p_j = b_jt| \sum w_jU_j$

In Section 3.3, we establish that the weighted number of tardy jobs can be solved by a pseudo-polynomial dynamic programming algorithm in $O(n^2N^2)$ time. In this section, we show that this problem is NP-hard in the ordinary sense. For simplification, we assume that $t_0$ is equal to 1.

**Theorem 4.1.** Problem 1 $|p\text{-batch}; r_j = 1; p_j = b_jt| \sum w_jU_j$ is binary NP-hard.

**Proof.** We show the result by reducing the Subset Product Problem, which is NP-hard [10], to our problem in polynomial time.

The Subset Product Problem is defined as follows: Given a set $R = \{q_1, q_2, \ldots, q_n\}$ of $m$ positive integers such that $\prod_{j \in R} q_j = A^2$, is there a subset $R'$ of $R$ such that $\prod_{j \in R'} q_j = A$?

Given any instance of the Subset Product Problem, we construct the corresponding instance of the scheduling problem with $n = 2m$ jobs. For each $j$ ($j = 1, \ldots, m$), we define a light job $J_j$ with $b_j = A^2q_j - 1$, $w_j = \ln q_j$, and $d_j = A^{2+j+1}$, and a heavy job $J_{m+j}$ with $b_{m+j} = A^2 - 1$, $w_{m+j} = A + 1$, and $d_{m+j} = A^{2+j+1}$. Note that $J_j$ and $J_{m+j}$ have the same due date, for $j = 1, \ldots, m$. 
We first show that the reduction is of polynomial time. Under a binary encoding, obvious lower bounds on the size of the input for Subset Product Problem are \( m \) and \( \log_2 A \). For the scheduling instance, each processing time is bounded above by \( A^{2m+1} \). An upper bound on the total size of the processing times under a binary encoding is \( 2m(2m+1) \log_2 A \), which is polynomially bounded in \( m \) and \( \log_2 A \). Similarly, upper bounds on the total size of the weights and due dates under a binary encoding are \( 2m \log_2(A+1) \) and \( 2m(m^2 + m + 1) \log_2 A \), respectively, which are also polynomially bounded in \( m \) and \( \log_2 A \). Thus, our reduction is polynomial.

In the remainder of the proof, we show that Subset Product Problem has a solution if and only if there exists a schedule for the scheduling instance such that \( \sum_{j=1}^{n} w_j U_j / \ln A \). First, suppose that \( X \) and \( Y \) define a solution to the Set Product Problem. Consider a schedule with \( m + 1 \) batches that is constructed as follows. Each light job \( j \), for \( j = 1, \ldots, m \), is assigned to begin by \( B_j \) if \( j \in X \), and is assigned to batch \( B_{m+1} \) if \( j \in Y \). The heavy jobs \( J_{m+1}, \ldots, J_{2m} \) are assigned to batches \( B_1, \ldots, B_m \), respectively. By Lemma 3.1.3, the processing time of batch \( B_j \), for \( j = 1, \ldots, m \), is either \( C(B_{j-1})(A^{2j}q_i - 1) \) or \( C(B_{j-1})(A^{2j} - 1) \) depending on whether \( j \in X \) or not, where \( C(B_{j-1}) \) is the completion time of the batch \( B_{j-1} \). So \( C(B_j) = A^{2j} \prod_{i \in X} q_i = d_m + j \) for \( j = 1, \ldots, m \), and therefore, each heavy job and each light job \( j \) for \( j \in X \) are on time. Consequently, \( \sum_{j=1}^{n} w_j U_j / \ln A \).

Conversely, suppose that there exists a schedule with \( \sum_{j=1}^{n} w_j U_j / \ln A \). In such a schedule, all heavy jobs have to be on time. Hence, \( J_{m+1} \) has to be processed in batch \( B_1 \), and neither job \( J \) nor \( J_{m+j} \) with \( j > 1 \) can be processed together with it in \( B_1 \). According to \( J_{m+2} \), is processed in batch \( B_2 \), which cannot begin before time \( A^2 \), the earliest possible completion time of \( B_1 \). Since \( J_{m+2} \) to be completed by its due date \( A^2 \), neither job \( J \) nor \( J_{m+j} \) with \( j > 2 \) can be processed together with it in \( B_2 \). Further, the earliest completion time of \( B_2 \) is \( A^6 \). So job \( J \) is tardy if it is processed in \( B_2 \). Repeating this line of reasoning, we deduce that each heavy job \( J_{m+j} \) and each on-time light job \( J \) are assigned to batch \( B_j \), for \( j = 1, \ldots, m \). Moreover, we assume without loss of generality that each tardy job \( J \) is assigned to batch \( B_{m+1} \). If job \( J \) is assigned to batch \( B_j \), then \( p(B_j) = C(B_{j-1})(A^{2j}q_i - 1) \); otherwise, \( p(B_j) = C(B_{j-1})(A^{2j} - 1) \). Let \( X \) and \( Y \) denote the sets of indices \( j = 1, \ldots, m \) for which \( J_j \in B_j \) and \( J_j \notin B_j \), respectively. To ensure that job \( J_{m+2} \) is on time, we require that \( C(B_m) = A^{2m+2} \prod_{i \in X} q_i \leq A^{2(m+1)} \). Thus, \( \prod_{i \in X} q_i \leq A \). The condition \( \sum_{j=1}^{n} w_j U_j / \ln A \) implies that \( \prod_{i \in Y} q_i \leq A \), or equivalently \( \prod_{i \in X} q_i \geq A \). Therefore, \( \prod_{i \in X} q_i = A \), which shows that \( X \) and \( Y \) define a solution to the Subset Product Problem.

5. Conclusions

In this paper, we study several single machine parallel-batch scheduling problems in which the processing time of a job is a linear function of its starting time. We give a polynomial-time algorithm for minimizing the maximum cost, an \( O(n^3) \) time algorithm for minimizing the number of tardy jobs, and an \( O(n^2) \) time algorithm for minimizing the total weighted completion time. Furthermore, we prove that the problem for minimizing the weighted number of tardy jobs is NP-hard. Note that the complexities of these problems are similar to those of the corresponding problems without deterioration. For the further research, it is interesting to resolve the complexity of problem 1|\( \text{p-batch} \); \( r_j = t_0 \); \( p_j = b_j t \) | \( \sum T_j \). Other extensions contain a study under general linear deterioration and the problems with the bounded model, i.e., \( b < n \).

Acknowledgements

The authors would like to thank the referees for their helpful comments and suggestions to improve the presentation of the paper.

References